Astronomical Data Analysis Software and Systems XVII **O4a.3**
ASP Conference Series, Vol. 394, © 2008
R. W. Argyle, P. S. Bunclark, and J. R. Lewis, eds.

Cloudspace: Virtual Environments in the VO

Matthew J. Graham and Roy D. Williams

California Institute of Technology, 770 S. Wilson Avenue, Pasadena, CA, 91125, USA

Abstract. The grid community is moving towards providing on-demand computing in the form of virtual workspaces - abstracted execution environments that are dynamically made available to authorized clients. In part this is a reaction to market forces represented by such commercial initiatives as Amazon EC2 and in part a solution to hot service deployment. One danger, though, is that a multiplicity of implementations will lead to a lack of interoperability. Such a concern in the VO regarding distributed data storage led to the development of VOSpace, a lightweight abstraction layer that sits on top of existing storage solutions such as SRB. In this paper, we introduce Cloudspace, a resource-oriented extension of VOSpace, that incorporates UWS, the VO pattern for managing asynchronous services, to form a natural habitat for virtual environments in the VO. A notable feature of the Cloudspace concept is that distributed data and computing can be managed seamlessly through a single mechanism thus making the astronomer's life easier as we move into a new era of sophisticated computational astronomy.

1. Introduction

For the good of the community, we decide to expose our world-beating data mining algorithm as a web service running on the local compute cluster in our basement. Initially the performance metrics are excellent as no-one is using it but as our service's reputation begins to grow, the local cluster is sometimes hard pressed. When a particularly apposite data set is released, however, the metrics go through the floor as the machines are thrashed to death by too many users.

We could react to this with the traditional solution of throwing more hardware at the problem but this normally carries the additional expense of porting the software to a (hopefully only slightly) different platform. When the algorithm is complex, this can be onerous and time-consuming and there is also now the added expense of trying to maintain consistency in a heterogeneous environment. Further expansions to meet usage will only serve to make this problem worse.

Ideally we would just like to be able to call up computing resources as and when we need them. Such dynamic allocation to meet the desires of more users, data and/or jobs is known as *on-demand* or *utility computing*. If the operational details of our algorithm in terms of its infrastructure requirements could also be abstracted in some way then we would not have to worry about providing multiple ports of our code.

2. Virtualization

Virtual workspaces (or environments) are an abstraction of an execution environment that can be served on demand to authorized clients using well-defined protocols. Resource quotas (e.g. CPU and memory share requirements) and software configurations (e.g. O/S and provided libraries and services) become metadata attached to a particular service virtualization.

One particular implementation of a virtualization is the *virtual machine* (VM), being the abstraction of a physical host machine. Our algorithm can be converted into a VM implementation (known as an *image*) and then deployed as multiple *instances* on any hardware which is running an appropriate *hypervisor* - this intercepts and emulates instructions from VMs to the underlying inftrastructure and also manages VM instances. VMWare[1] and Xen[2] are popular examples of VMs.

We are still reliant on someone providing the computing resources to host our VMs but commercial enterprises such as Amazon EC2[3] (Elastic Compute Cloud) and FlexiScale[4] strongly suggest that this will not be an issue. In fact as more providers (such Google, IBM and Microsoft) jump on the bandwagon, we will be faced with the new problem of ensuring that we are using the right sort of VM for the hardware provider's hypervisor: for example, EC2 uses the Xen hypervisor but there is no guarantee that other providers will follow suit.

3. Hypervisor abstraction

Obviously we can solve this with another abstraction layer that hides the details of the variety of underlying hypervisor implementations. The Globus WorkspaceService[5] is an example of such a layer. Workspaces (VMs) are described in terms of metadata instances (containing a pointer to the VM image as well as configuration information such as networking) and deployment requests (specifying what resources, e.g. deployment time, CPUs, memory, etc. should be assigned to the VM). Authorized workspace instances are created by a workspace factory service deployed on the computing hardware. A prototype system has been developed by the GLobus team employing both the University of Chicago Teraport cluster and Amazon EC2 as hardware, although the user never has any idea on which platform their VM is running.

Unfortunately Globus solutions have a reputation for being complex and heavyweight: for example, you tend to need a full Globus installation on the client as well as the server side. In consequence, we propose an alternate lightweight abstraction layer called *Cloudspace*. This derives from our work on VOSpace (Graham, Morris & Rixon 2007), a lightweight abstraction layer

---

[1] `http://www.vmware.com`

[2] `http://www.xensource.com`

[3] `http://aws.amazon.com/ec2`

[4] `http://www.flexiscale.com`

[5] `http://workspace.globus.org`

for accessing distributed data storage solutions in the VO. Cloudspace is resource oriented, treats data and services both as first-class entities and employs the IVOA *Universal Worker Service* (UWS, Rixon 2007) interface to manage services.

4. Resource orientation

Resource-oriented services are an alternate to the RPC-oriented approach associated with most SOAP-based services. A *resource* is an abstract set of information such as a play by Shakespeare, and each resource may be identified by one or more *logical identifiers*, such as the Tempest or Shakespeare's last play. A logical identifier may be resolved to a physical *resource representation*, such as a paper copy of the Tempest or a DVD of a performance, and the process of obtaining a physical resource representation is *computation*. Resource representations are immutable - if you alter the words in the paper copy then it is no longer a valid representation of the Tempest - but one representation can be transformed into another in an isomorphic and lossless way, for example, someone can make an audio recording of the play from the text version. The results of any computation are also a resource.

5. Identifiers and representations

All objects (resources) within Cloudspace are identified by one or more URIs with different schemes returning different representations (see Table 1). This is in contrast to the more common approach of using MIME types and HTTP headers to determine which representation to return.

Table 1. URI schemes and representations returned for Cloudspace entities

	Data object	Service image	Service instance
URI scheme	vos://	ivo://	csp://
Representation	VOSpace <node>	VOResource	UWS

The VOSpace <node> construct describes any arbitrary metadata associated with the data object in terms of properties and also what transformations are supported. The VOResource representation (Plante et al. 2007) specifies the service's metadata including its virtualization metadata. The exact details still need to be defined although this will probably be based on the Globus Workspace data model. The UWS representation includes information about what state the particular service instance is in - pending, queued, executing, etc. -, when the service might terminate and what the results are.

6. Computing in the cloud

Let's imagine that we have a VM image of our service, identified in the VO registry as ivo://nvo.caltech/MyService. First we upload the actual bytes of the image to a data node in Cloudspace using the VOSpace identifier: vos://nvo.caltech!vospace/MyService. We can create a service instance from the image by sending a HTTP POST request to the image's UWS URI: csp://nvo.caltech/services/MyService. A service instance will now exist (csp://nvo.caltech/services/MyService/1234) in a pending state. To commit this instance to execution, we would post a request to csp://nvo.caltech/services/MyService/1234/quote.

We could now call this service instance using a standard service endpoint such as http://nvo.caltech/services/MyService/1234 and pass it arguments. However, if the data and parameter files that we want the service to use are already in Cloudspace then we can actually define the computation in terms of component URIs using the comp:// scheme, for example:

```
comp://nvo.caltech/service/MyService/1234+
     data@vos://nvo!caltech/vospace/myTable1+
     params@vos://nvo!caltech/vospace/myParam1
```

This can also be mapped to a data object, e.g. vos://nvo!caltech/vospace/myResult1, which holds the specific results of the computation. Such an approach allows *memoization* - data caching a function call so that the results can be quickly served to a subsequent equivalent function call.

7. Proof of concept

We are implementing a proof-of-concept system to demonstrate Cloudspace using the NetKernel[6] resource-oriented application server and Ruby scripts to interface with a VM deployed on Amazon EC2. In principle, it should be straightforward enough to extend Cloudspace so that it can work with other packaged services, e.g. war files, and not just VMs.

References

Graham, M. J. Morris, D. Rixon, G. 2007, in ASP Conf. Ser. 376, ADASS XVI, ed. R. A. Shaw, F. Hill, & D. J. Bell (San Francisco: ASP), 567

Plante, R., et al. 2007, VOResource 1.02, http://www.ivoa.net/Documents/latest/VOResource.html

Rixon, G. 2007, Universal Worker Server v0.3, http://www.ivoa.net/internal/IVOA/IvoaGridAndWebServices/UWS-0.3.pdf

[6] `http://www.1060.org`